Incremental, Inductive Coverability

Johannes Kloos, Rupak Majumdar, Filip Niksic, and Ruzica Piskac

MPI-SWS

Abstract. We give an incremental, inductive (IC3) procedure to check coverability of well-structured transition systems. Our procedure generalizes the IC3 procedure for safety verification that has been successfully applied in finite-state hardware verification to infinite-state well-structured transition systems. We show that our procedure is sound, complete, and terminating for *downward-finite* well-structured transition systems —where each state has a finite number of states below it— a class that contains extensions of Petri nets, broadcast protocols, and lossy channel systems.

We have implemented our algorithm for checking coverability of Petri nets. We describe how the algorithm can be efficiently implemented without the use of SMT solvers. Our experiments on standard Petri net benchmarks show that IC3 is competitive with state-of-the-art implementations for coverability based on symbolic backward analysis or expand-enlarge-and-check algorithms both in time taken and space usage.

1 Introduction

The IC3 algorithm [3] was recently introduced as an efficient technique for safety verification of hardware. It computes an inductive invariant by maintaining a sequence of over-approximations of forward-reachable states, and incrementally strengthening them based on counterexamples to inductiveness. The counterexamples are obtained using a backward exploration from error states. Efficient implementations of the procedure show remarkably good performance on hardware benchmarks [8].

A natural direction is to extend the IC3 algorithm to classes of systems beyond finite-state hardware circuits. Indeed, an IC3-like technique was recently proposed for interpolation-based software verification [5], and the technique was generalized to finite-data pushdown systems as well as systems using linear real arithmetic [15]. Hoder and Bjørner show that their generalized IC3 procedure terminates on timed pushdown automata [15], and it is natural to ask for what other classes of infinite-state systems does IC3 form a decision procedure for safety verification.

In this paper, we consider well-structured transition systems (WSTS) [1,12]. WSTS are infinite-state transition systems whose states have a well-quasi order, and whose transitions satisfy a monotonicity property w.r.t. the quasi-order. WSTS capture many important infinite-state models such as Petri nets and their

monotonic extensions [11,4,7,13], broadcast protocols [9,10], and lossy channel systems [2]. A general decidability result shows that the coverability problem (reachability in an upward-closed set) is decidable for WSTS [1]. The decidability result performs a backward reachability analysis, and shows, using properties of well-quasi orderings, that the reachability procedure must terminate. In many verification problems, techniques based on computing inductive invariants outperform methods based on backward or forward reachability analysis; indeed, IC3 for hardware circuits is a prime example. Thus, it is natural to ask if there is a IC3-style decision procedure for coverability analysis for WSTS.

We answer this question positively. We give a generalization of IC3 for WSTS, and show that it terminates on the class of *downward-finite* WSTS, in which each state has a finite number of states lower than itself. The class of downward-finite WSTS contains the most important classes of WSTS used in verification, including Petri nets and their extensions, broadcast protocols, and lossy channel systems. Hence, our results show that IC3 is a decision procedure for the coverability problem for these classes of systems. While termination is trivial in the finite-state case, our technical contribution is to show, using the termination of the backward reachability procedure, that the sequence of (downward closed) invariants produced by IC3 is guaranteed to converge. We also show that the assumption of downward-finiteness is necessary: we give a (technical) example of a general WSTS on which the algorithm does not terminate.

We have implemented our algorithm in a tool called IIC to check coverability in Petri nets. Using combinatorial properties of Petri nets, we derive an optimized implementation of the algorithm that does not use an SMT solver. Our implementation shows that IIC outperforms several state-of-the-art implementations of coverability [13,16] on a set of Petri net examples, both in space and in time requirements. For example, on a set of standard Petri net examples, we outperform implementations of EEC and backward reachability, often by orders of magnitude.

2 Preliminaries

Well-quasi Orders For a set X, a relation $\preceq \subseteq X \times X$ is a *well-quasi-order (wqo)* if it is reflexive, transitive, and if for every infinite sequence $x_0, x_1, \ldots$ of elements from X, there exists $i < j$ such that $x_i \preceq x_j$. A set $Y \subseteq X$ is *upward-closed* if for every $y \in Y$ and $x \in X$, $y \preceq x$ implies $x \in Y$. Similarly, a set $Y \subseteq X$ is *downward-closed* if for every $y \in Y$ and $x \in X$, $x \preceq y$ implies $x \in Y$. For a set Y, by $Y\!\uparrow$ we denote its upward closure, i.e., the set $\{x \mid \exists y \in Y, y \preceq x\}$. For a singleton $\{x\}$, we simply write $x\!\uparrow$ for $\{x\}\!\uparrow$. Similarly, we define $Y\!\downarrow = \{x \mid \exists y \in Y, x \preceq y\}$ for the downward closure of a set Y. Clearly, $Y\!\uparrow$ (resp., $Y\!\downarrow$) is an upward-closed set (resp. downward-closed) for each Y. The union and intersection of upward-closed sets are upward-closed, and the union and intersection of downward-closed sets are downward-closed. Furthermore, the complement of an upward-closed set is downward-closed, and the complement of a downward-closed set is upward-closed. For the convenience of the reader, we will mark upward-closed sets with a

small up-arrow superscript, like this: $U^{\uparrow}$, and downward-closed sets with a small down-arrow superscript, like this: $D^{\downarrow}$.

A basis of an upward-closed set Y is a set $Y_b \subseteq Y$ such that $Y = \bigcup_{y \in Y_b} y\!\uparrow$. It is known [14,1,12] that any upward-closed set Y in a wqo has a finite basis: the set of minimal elements of Y has finitely many equivalence classes under the equivalence relation $\preceq \cap \succeq$, so take any system of representatives. We write $\min Y$ for such a system of representatives. Moreover, it is known that any non-decreasing sequence $I_0 \subseteq I_1 \subseteq \ldots$ of upward-closed sets eventually stabilizes, i.e., there exists $k \in \mathbb{N}$ such that $I_k = I_{k+1} = I_{k+2} = \ldots$.

A wqo $(X, \preceq)$ is *downward-finite* if for each $x \in X$, the downward closure $x\!\downarrow$ is a finite set.

Examples: Let $\mathbb{N}^k$ be the set of k-tuples of natural numbers, and let $\preceq$ be pointwise comparison: $v \preceq v'$ if $v_i \leqslant v'_i$ for $i = 1, \ldots, k$. Then, $(\mathbb{N}^k, \preceq)$ is a downward-finite wqo [6].

Let A be a finite alphabet, and consider the subword ordering $\preceq$ on words over A, given by $w \preceq w'$ for $w, w' \in A^*$ if w results from w' by deleting some occurrences of symbols. Then $(A^*, \preceq)$ is a downward-finite wqo [14].

Well-structured Transition Systems A well-structured transition system (WSTS) $(\Sigma, I, \rightarrow, \preceq)$ consists of a set Σ of states, a finite set $I \subseteq \Sigma$ of initial states, a transition relation $\rightarrow \subseteq \Sigma \times \Sigma$, and a well-quasi ordering $\preceq \subseteq \Sigma \times \Sigma$ such that for all $s_1, s_2, t_1 \in \Sigma$ such that $s_1 \rightarrow s_2$ and $s_1 \preceq t_1$ there exists t_2 such that $t_1 \rightarrow^* t_2$ and $s_2 \preceq t_2$. A WSTS is downward-finite if $(\Sigma, \preceq)$ is downward-finite.

Let $x, y \in \Sigma$. If $x \rightarrow y$, we call x a *predecessor* of y, and y a *successor* of x. We write $\mathrm{pre}(x) := \{y \mid y \rightarrow x\}$ for the *set of predecessors* of x, and $\mathrm{post}(x) := \{y \mid x \rightarrow y\}$ for the *set of successors* of x. For $X \subseteq \Sigma$, $\mathrm{pre}(X)$ and $\mathrm{post}(X)$ are defined as natural extensions, i.e., $\mathrm{pre}(X) = \bigcup_{x \in X} \mathrm{pre}(x)$ and $\mathrm{post}(X) = \bigcup_{x \in X} \mathrm{post}(x)$.

We write $x \rightarrow^k y$ if there are states $x_0, \ldots, x_k \in \Sigma$ such that $x_0 = x$, $x_k = y$ and $x_i \rightarrow x_{i+1}$ for $0 \leq i < k$. Furthermore, $x \rightarrow^* y$ iff there exists a $k \geqslant 0$ such that $x \rightarrow^k y$, i.e., $\rightarrow^*$ is the reflexive and transitive closure of $\rightarrow$. We say that there is a *path from x to y of length k* if $x \rightarrow^k y$, and that there is a path from x to y if $x \rightarrow^* y$.

The set of *k-reachable* states Reach_k is the set of states reachable in at most k steps, formally, $\mathsf{Reach}_k := \{y \in \Sigma \mid \exists k' \leqslant k, \exists x \in I, x \rightarrow^{k'} y\}$. The set of *reachable* states $\mathsf{Reach} := \bigcup_{k \geq 0} \mathsf{Reach}_k = \{y \mid \exists x \in I, x \rightarrow^* y\}$. Using downward closure, we can define the *k-th cover* Cover_k and the *cover* Cover of the WSTS as $\mathsf{Cover}_k := \mathsf{Reach}_k\!\downarrow$ and $\mathsf{Cover} := \mathsf{Reach}\!\downarrow$. The *coverability problem for WSTS* asks, given a WSTS $(\Sigma, I, \rightarrow, \preceq)$ and a downward-closed set $P^{\downarrow}$, if every reachable state is contained in $P^{\downarrow}$, i.e., if $\mathsf{Reach} \subseteq P^{\downarrow}$. It is easy to see that this question is equivalent to checking if $\mathsf{Cover} \subseteq P^{\downarrow}$.

In the following, we make some standard effectiveness assumptions on WSTS [1,12]. We assume that $\preceq$ is decidable, and that for any state $x \in \Sigma$, there is a computable procedure that returns a finite basis for $\mathrm{pre}(x\!\uparrow)$. These assumptions are met by most classes of WSTS considered in verification [12].

Under the preceding effectiveness assumptions, one can show that the coverability problem is decidable for WSTS by a backward-search algorithm [1]. The main construction is the following sequence of upward-closed sets:

$$\mathsf{U}^{\uparrow}{}_0 := \Sigma \setminus P^{\downarrow}\,, \qquad \mathsf{U}^{\uparrow}{}_{i+1} := \mathsf{U}^{\uparrow}{}_i \cup \mathrm{pre}(\mathsf{U}^{\uparrow}{}_i)\,. \qquad \text{(BackwardReach)}$$

It is easy to see that the sequence of sets $\mathsf{U}^{\uparrow}{}_i$ forms an increasing chain of upward-closed sets, therefore it eventually stabilizes: there is some L such that $\mathsf{U}^{\uparrow}{}_L = \mathsf{U}^{\uparrow}{}_{L+i}$ for all $i \geq 0$. Then, $\mathsf{Cover} \subseteq P^{\downarrow}$ iff $I \cap \mathsf{U}^{\uparrow}{}_L = \emptyset$. Moreover, if $I \cap \mathsf{U}^{\uparrow}{}_L = \emptyset$, then $\Sigma \setminus \mathsf{U}^{\uparrow}{}_L$ contains I, is contained in $P^{\downarrow}$ and satisfies $\mathrm{post}(\Sigma \setminus \mathsf{U}^{\uparrow}{}_L) \subseteq \Sigma \setminus \mathsf{U}^{\uparrow}{}_L$.

We generalize from $\Sigma \setminus \mathsf{U}^{\uparrow}{}_L$, in the style of inductive invariants for safety verification, to the notion of an (inductive) *covering set*. A downward-closed set $C^{\downarrow}$ is called a *covering set* for $P^{\downarrow}$ iff (a) $I \subseteq C^{\downarrow}$, (b) $C^{\downarrow} \subseteq P^{\downarrow}$, and (c) if $\mathrm{post}(C^{\downarrow}) \subseteq C^{\downarrow}$. By induction, it is clear that $\mathsf{Cover} \subseteq C^{\downarrow} \subseteq P^{\downarrow}$ for any covering set $C^{\downarrow}$. Therefore, to solve the coverability problem, it is sufficient to exhibit any covering set.

3 IC3 for Coverability

We now describe an algorithm for the coverability problem that takes as input a WSTS $(\Sigma, I, \to, \preceq)$ and a downward-closed set $P^{\downarrow}$, and constructs either a path from some state in I to a state not in $P^{\downarrow}$ (if $\mathsf{Cover} \not\subseteq P^{\downarrow}$), or an inductive covering set for $P^{\downarrow}$. In the algorithm we consider sets that are not necessarily inductive by themselves, but they are *inductive relative to* some other sets. Formally, for a set $R^{\downarrow}$ such that $I \subseteq R^{\downarrow}$, a downward-closed set $S^{\downarrow}$ is inductive relative to $R^{\downarrow}$ if $I \subseteq S^{\downarrow}$ and $\mathrm{post}(R^{\downarrow} \cap S^{\downarrow}) \subseteq S^{\downarrow}$. An upward-closed set $U^{\uparrow}$ is inductive relative to $R^{\downarrow}$ if its downward-closed complement $\Sigma \setminus U^{\uparrow}$ is inductive relative to $R^{\downarrow}$, i.e. if $I \cap U^{\uparrow} = \emptyset$ and $\mathrm{post}(R^{\downarrow} \setminus U^{\uparrow}) \subseteq \Sigma \setminus U^{\uparrow}$.

It can be easily shown that the condition $\mathrm{post}(R^{\downarrow} \cap S^{\downarrow}) \subseteq S^{\downarrow}$ is equivalent to $\mathrm{pre}(\Sigma \setminus S^{\downarrow}) \cap R^{\downarrow} \cap S^{\downarrow} = \emptyset$. Stated in terms of an upward-closed set $U^{\uparrow}$, the equivalent condition is $\mathrm{pre}(U^{\uparrow}) \cap R^{\downarrow} \setminus U^{\uparrow} = \emptyset$. Since all these conditions are equivalent, we will use them interchangeably.

3.1 Algorithm

Figure 1 shows the algorithm as a set of non-deterministic state transition rules, similar to [15]. A state of the computation is either the initial state Init, the special states valid and $\mathsf{invalid}$ that denote termination, or a pair $\mathbf{R} \mid Q$ defined as follows.

The first component of the pair is a vector $\mathbf{R}$ of downward-closed sets, indexed starting from 0. The elements of $\mathbf{R}$ are denoted $R_i^{\downarrow}$. In particular, we denote by $R_0^{\downarrow}$ the downward closure of I, i.e., $R_0^{\downarrow} = I{\downarrow}$. These sets contain the successive approximations to the inductive covering set. The function length gives the length of the vector, disregarding $R_0^{\downarrow}$, i.e., $\mathrm{length}(R_0^{\downarrow}, \ldots, R_N^{\downarrow}) = N$. If it is clear from the context which vector is meant, we often abbreviate $\mathrm{length}(\mathbf{R})$

$$[\mathsf{Initialize}]\frac{}{\mathsf{Init} \mapsto I\downarrow \mid \varnothing} \qquad [\mathsf{CandidateNondet}]\frac{a \in R_N^{\downarrow} \setminus P^{\downarrow}}{\mathbf{R} \mid \varnothing \mapsto \mathbf{R} \mid \langle a, N\rangle}$$

$$[\mathsf{ModelSyn}]\frac{\min Q = \langle a, 0\rangle}{\mathbf{R} \mid Q \mapsto \mathsf{invalid}} \qquad [\mathsf{ModelSem}]\frac{\min Q = \langle a, i\rangle \quad I \cap a\uparrow \neq \varnothing}{\mathbf{R} \mid Q \mapsto \mathsf{invalid}}$$

$$[\mathsf{DecideNondet}]\frac{\min Q = \langle a, i\rangle \quad i > 0 \quad b \in \mathrm{pre}(a\uparrow) \cap R_{i-1}^{\downarrow} \setminus a\uparrow}{\mathbf{R} \mid Q \mapsto \mathbf{R} \mid Q.\textsc{Push}(\langle b, i-1\rangle)}$$

$$[\mathsf{Conflict}]\frac{\min Q = \langle a, i\rangle \quad i > 0 \quad \mathrm{pre}(a\uparrow) \cap R_{i-1}^{\downarrow} \setminus a\uparrow = \varnothing \quad b \in \mathrm{Gen}_{i-1}(a)}{\mathbf{R} \mid Q \mapsto \mathbf{R}[R_k^{\downarrow} \leftarrow R_k^{\downarrow} \setminus b\uparrow]_{k=1}^{i} \mid Q.\textsc{PopMin}}$$

$$[\mathsf{Induction}]\frac{R_i^{\downarrow} = \Sigma \setminus \{r_{i,1}, \ldots, r_{i,m}\}\uparrow \quad b \in \mathrm{Gen}_i(r_{i,j}) \text{ for some } 1 \leq j \leq m}{\mathbf{R} \mid \varnothing \mapsto \mathbf{R}[R_k^{\downarrow} \leftarrow R_k^{\downarrow} \setminus b\uparrow]_{k=1}^{i+1} \mid \varnothing}$$

$$[\mathsf{Valid}]\frac{R_i^{\downarrow} = R_{i+1}^{\downarrow} \text{ for some } i < N}{\mathbf{R} \mid Q \mapsto \mathsf{valid}} \qquad [\mathsf{Unfold}]\frac{R_N^{\downarrow} \subseteq P^{\downarrow}}{\mathbf{R} \mid \varnothing \mapsto \mathbf{R} \cdot \Sigma \mid \varnothing}$$

Fig. 1. The rule system for a IC3-style algorithm for WSTS – generic version. The map Gen_i is defined in equation (1).

simply with N. We write $\mathbf{R} \cdot X$ for the concatenation of the vector $\mathbf{R}$ with the downward closed set X: $(R_0^{\downarrow}, \ldots, R_N^{\downarrow}) \cdot X = (R_0^{\downarrow}, \ldots, R_N^{\downarrow}, X)$.

The second component of the pair is a priority queue Q, containing elements of the form $\langle a, i\rangle$, where $a \in \Sigma$ is a state and $i \in \mathbb{N}$ is a natural number. The priority of the element is given by i, and is called the level of the element. The statement $\langle a, i\rangle \in Q$ means that the priority queue contains an element of the given form, while $\min Q = \langle a, i\rangle$ means that the minimal element of the priority queue has the given form. Furthermore, $Q.\textsc{PopMin}$ yields Q after removal of its minimal element, and $Q.\textsc{Push}(x)$ yields Q after adding element x.

The elements of Q are states that lead outside of $P^{\downarrow}$. Let $\langle a, i\rangle$ be an element of Q. Either a is a state that is in R_i and outside of $P^{\downarrow}$, or there is a state b leading to ${P^{\downarrow}}^c$ such that $a \in \mathrm{pre}(b\uparrow)$. Our goal is to try to discard those states and show that they are not reachable from the initial state, as R_i denotes an overapproximation of the states reachable in i or less steps. If an element of Q is reachable from the initial state, then $\mathsf{Cover} \not\subseteq P^{\downarrow}$.

The state valid signifies that the search has terminated with the result that $\mathsf{Cover} \subseteq P^{\downarrow}$ holds, while $\mathsf{invalid}$ signifies that the algorithm has terminated with the result that $\mathsf{Cover} \not\subseteq P^{\downarrow}$. In the description of the algorithm, we will omit the actual construction of certificates and instead just state that the algorithm terminates with $\mathsf{invalid}$ or valid; the calculation of certificates is straightforward.

The transition rules of the algorithm are of the form

$$[\text{Name}]\frac{C_1 \quad \cdots C_k}{\sigma \mapsto \sigma'} \tag{Rule}$$

and can be read thus: whenever the algorithm is in state σ and conditions C_1–C_k are fulfilled, the algorithm can apply rule [Name] and transition to state σ'. We write $\sigma \mapsto \sigma'$ if there is some rule such that the algorithm applies the rule to go from σ to σ'. We write $\mapsto^*$ for the reflexive transitive closure of $\mapsto$.

Let Inv be a predicate on states. When we say that a rule *preserves the invariant* Inv if whenever σ satisfies Inv and conditions C_1 to C_k are met, it also holds that σ' satisfies Inv.

Two of the rules use the map $\mathrm{Gen}_i : \Sigma \to 2^{\Sigma}$. It yields those states that are valid generalizations of a relative to some set $R_i^{\downarrow}$. A state b is a generalization of the state a relative to the set $R_i^{\downarrow}$, if $b \preceq a$ and $b\!\uparrow$ is inductive relative to $R_i^{\downarrow}$. Formally,

$$\mathrm{Gen}_i(a) := \{b \mid b \preceq a \wedge b\!\uparrow \cap I = \varnothing \wedge \mathrm{pre}(b\!\uparrow) \cap R_i^{\downarrow} \setminus b\!\uparrow = \varnothing\} \tag{1}$$

Finally, the notation $\mathbf{R}[R_k^{\downarrow} \leftarrow R'^{\downarrow}_k]_{k=1}^i$ means that $\mathbf{R}$ is transformed by replacing $R_k^{\downarrow}$ by $R'^{\downarrow}_k$ for each $k = 1, \dots, i$, i.e.,

$$\mathbf{R}[R_k^{\downarrow} \leftarrow R'^{\downarrow}_k]_{k=1}^i = (R_0^{\downarrow}, R'^{\downarrow}_1, \dots, R'^{\downarrow}_i, R_{i+1}^{\downarrow}, \dots, R_n^{\downarrow}).$$

We provide an overview of each rule of the calculus. [Initialize] The algorithm starts by defining the first downward-closed set $R_0^{\downarrow}$ to be the downward closure of the initial state.

[CandidateNondet] If there is a state a such that $a \in R_N^{\downarrow}$ but at the same time it is not an element of $P^{\downarrow}$ we add $\langle a, N\rangle$ to the priority queue Q.

[DecideNondet] To check if the elements of Q are spurious counterexamples, we start by processing an element a with the lowest level i. If there is an element b in $R_{i-1}^{\downarrow}$ such that $b \in \mathrm{pre}(a\!\uparrow)$, then we add $\langle b, i-1\rangle$ to the priority queue Q.

[ModelSyn] If the queue contains a state a from the level 0, then we have found a counterexample trace and the algorithm terminates in the state invalid.

[ModelSem] Similarly, if the queue contains a state a such $I \cap a\!\uparrow \neq \emptyset$, this is again a counterexample trace and the algorithm terminates in the state invalid.

[Conflict] If none of predecessors of a state a from the level i is contained in $R_{i-1}^{\downarrow} \setminus a\!\uparrow$, then a belongs to a spurious counterexample trace and can be safely removed from the queue. Additionally, we update the downward-closed sets $R_1^{\downarrow}, \dots, R_i^{\downarrow}$ as follows: since the states in $a\!\uparrow$ are not reachable in i steps, they can be safely removed from all the sets $R_1^{\downarrow}, \dots, R_i^{\downarrow}$. Moreover, instead of $a\!\uparrow$ we can remove even a bigger set $b\!\uparrow$, for any state b which is a generalization of the state a relative to $R_{i-1}^{\downarrow}$, as defined in (1).

[Induction] If for some state $r_{i,j}\!\uparrow$ that was previously removed from $R_i^{\downarrow}$, a set $\Sigma \setminus r_{i,j}\!\uparrow$ becomes inductive relative to $R_i^{\downarrow}$ (i.e. $\mathrm{post}(R_i^{\downarrow} \cap r_{i,j}\!\downarrow) \subseteq r_{i,j}\!\downarrow$), none of the states in $r_{i,j}\!\uparrow$ can be reached in at most $i+1$ steps. Thus, we can safely remove $r_{i,j}\!\uparrow$ from $R_{i+1}^{\downarrow}$ as well. Similarly as in [Conflict], we can even remove $b^{\uparrow}$ for any generalization $b \in \mathrm{Gen}_i(r_{i,j})$.

[Valid] If there is a downward-closed set $R_i^{\downarrow}$ such that $R_i^{\downarrow} = R_{i+1}^{\downarrow}$, the algorithm terminates in the state valid.

[Unfold] If the queue is empty and all elements of $R^{\downarrow}_N$ are in $P^{\downarrow}$, we start with a construction of the next set $R^{\downarrow}_{N+1}$. Initially, $R^{\downarrow}_{N+1}$ contains all the states, $R^{\downarrow}_{N+1} = \Sigma$, and we append $R^{\downarrow}_{N+1}$ to the vector $\mathbf{R}$.

3.2 Soundness

We first show that the algorithm is sound: if it terminates, it produces the right answer. If it terminates in the state invalid there is a path from an initial state to a state outside of $P^{\downarrow}$, and if it terminates in the state valid then $\mathsf{Cover} \subseteq P^{\downarrow}$.

We prove soundness by showing that on each state $\mathbf{R} \mid Q$ the following invariants are preserved by the transition rules:

$$I \subseteq R^{\downarrow}_i \qquad \text{for all } 0 \leq i \leq N \tag{I1}$$

$$\mathrm{post}(R^{\downarrow}_i) \subseteq R^{\downarrow}_{i+1} \qquad \text{for all } 0 \leq i < N \tag{I2}$$

$$R^{\downarrow}_i \subseteq R^{\downarrow}_{i+1} \qquad \text{for all } 0 \leq i < N \tag{I3}$$

$$R^{\downarrow}_i \subseteq P^{\downarrow} \qquad \text{for all } 0 \leq i < N \tag{I4}$$

These properties imply $R^{\downarrow}_i \supseteq \mathsf{Cover}_i$, that is, the region R_i provides an over-approximation of the i-cover.

The first step of the algorithm (rule [Initialize]) results with the state $I{\downarrow} \mid \varnothing$, which satisfies (I2)–(I4) trivially, and $I \subseteq I{\downarrow}$ establishes (I1). The following lemma states that the invariants are preserved by rules that do not result in valid or invalid. For lack of space, full proofs are given in Appendix A.

Lemma 1. *The rules* [Unfold], [Induction], [Conflict], [CandidateNondet], *and* [DecideNondet] *preserve* (I1) – (I4),

By induction on the length of the trace, it can be shown that if $\mathsf{Init} \mapsto^* \mathbf{R} \mid Q$, then $\mathbf{R} \mid Q$ satisfies (I1) – (I4). When $\mathsf{Init} \mapsto^* \mathsf{valid}$, there is a state $\mathbf{R} \mid Q$ such that $\mathsf{Init} \mapsto^+ \mathbf{R} \mid Q \mapsto \mathsf{valid}$, and the last applied rule is [Valid]. To be able to apply [Valid], there is an i such that $R^{\downarrow}_i = R^{\downarrow}_{i+1}$.

We claim that $R^{\downarrow}_i$ is an inductive covering set. This claim follows from the fact that (1) $R^{\downarrow}_i \subseteq P^{\downarrow}$ by invariant (I4), (2) $I \subseteq R^{\downarrow}_i$ by invariant (I1), and (3) $\mathrm{post}(R^{\downarrow}_i) \subseteq R^{\downarrow}_{i+1} = R^{\downarrow}_i$ by invariant (I2). This claim proves the correctness of the algorithm in case $\mathsf{Cover} \subseteq P^{\downarrow}$:

Theorem 1. *[Soundness of coverability] Given a WSTS $(\Sigma, I, \rightarrow, \preceq)$ and a downward-closed set $P^{\downarrow}$, if $\mathsf{Init} \mapsto^* \mathsf{valid}$, then $\mathsf{Cover} \subseteq P^{\downarrow}$.*

We next consider the case when $\mathsf{Cover} \not\subseteq P^{\downarrow}$. The following lemma describes the structure of the priority queues used in the algorithm.

Lemma 2. *Let $\mathsf{Init} \mapsto^* \mathbf{R} \mid Q$. If $Q \neq \varnothing$, then for every $\langle a, i \rangle \in Q$, there is a path from a to some $b \in \Sigma \setminus P^{\downarrow}$.*

$$\text{[Candidate]}\ \frac{a \in R_N^{\downarrow} \cap D_0}{\mathbf{R} \mid \varnothing \mapsto \mathbf{R} \mid \langle a, N\rangle} \qquad \text{[Decide]}\ \frac{\min Q = \langle a, i\rangle \quad i > 0 \quad b \in D_{N-i+1} \cap R_{i-1}^{\downarrow} \quad b \to a}{\mathbf{R} \mid Q \mapsto \mathbf{R} \mid Q.\textsc{Push}(\langle b, i-1\rangle)}$$

Fig. 2. Rules replacing [CandidateNondet] and [DecideNondet] in Fig. 1.

Theorem 2. *[Soundness of uncoverability] Given a WSTS $(\Sigma, I, \to, \preceq)$ and a downward-closed set $P^{\downarrow}$, if $\mathsf{Init} \mapsto^* \mathsf{invalid}$, then $\mathsf{Cover} \not\subseteq P^{\downarrow}$.*

Proof. The assumption $\mathsf{Init} \mapsto^* \mathsf{invalid}$ implies that there is some state $\mathbf{R} \mid Q$ such that $\mathsf{Init} \mapsto^* \mathbf{R} \mid Q \mapsto \mathsf{invalid}$, and the last applied rule was either [ModelSyn] or [ModelSem]. In both cases $Q \neq \emptyset$.

If the last applied rule was [ModelSem], there is an $\langle a, i\rangle \in Q$ such that $a\!\uparrow \cap I \neq \emptyset$. By Lemma 2 there is a path from a to $b \in \Sigma \setminus P^{\downarrow}$. Let $a' \in a\!\uparrow \cap I$. Since $(\Sigma, I, \to, \preceq)$ is a WSTS, there is b' such that $a' \to^* b'$ and $b' \succeq b$. The set $\Sigma \setminus P^{\downarrow}$ is upward-closed, and thus $b' \in \Sigma \setminus P^{\downarrow}$. The path $a' \to^* b'$ is a path from I to $\Sigma \setminus P^{\downarrow}$, proving that $\mathsf{Cover} \not\subseteq P^{\downarrow}$.

If the last applied rule was [ModelSyn], then $\langle a, 0\rangle \in Q$. This implies $a \in R_0^{\downarrow} = I\!\downarrow$, as $R_0^{\downarrow}$ is constant in the algorithm. Equivalently, $a\!\uparrow \cap I \neq \varnothing$ and we apply the same arguments as in the case for [ModelSem]. ☐

3.3 Termination

While the above non-deterministic rules guarantee soundness for any WSTS, termination requires some additional choices. We modify the [DecideNondet] and [CandidateNondet] rules into more restricted rules [Decide] and [Candidate], while all other rules are unchanged. Figure 2 shows the new rules [Candidate] and [Decide]. These rules additionally use a sequence of sets D_i. Intuitively, there can be infinitely many elements in $R_N^{\downarrow} \setminus P^{\downarrow}$. Sets D_i provide a finite representation of those elements.

Recall the sequence $\mathsf{U}^{\uparrow}{}_i$ of backward reachable states from (BackwardReach). We define sets D_i using sets $\mathsf{U}^{\uparrow}{}_i$. The set D_i captures all new elements that are introduced in $\mathsf{U}^{\uparrow}{}_i$ and that were not present in the previous iterations. Formally, we define sets D_i as follows:

$$D_0 := \min(\Sigma \setminus P^{\downarrow}) \qquad\qquad D_{i+1} := \bigcup_{a \in D_i} \min(\mathrm{pre}(a\!\uparrow)) \setminus \mathsf{U}^{\uparrow}{}_i \ . \tag{2}$$

By induction, and the finiteness of the set of minimal elements, we have that D_i is finite for all $i \geqslant 0$. Further, assume that $\mathsf{U}^{\uparrow}{}_L = \mathsf{U}^{\uparrow}{}_{L+1}$. Then, $D_i = \varnothing$ for all $i > L$. As a consequence, the set $\bigcup_{i \geqslant 0} D_i$ is finite.

It is easy to show that the restricted rules still preserve the invariants (I1) – (I4), and thus the modified algorithm is still sound. From now, we focus on the modified algorithm.

To show that the algorithm always terminates, we first show that the system can make progress until some final state is reached.

Proposition 1 (Maximal finite sequences). *Let* $\mathsf{Init} = \sigma_0 \mapsto \sigma_1 \mapsto \cdots \mapsto \sigma_K$ *be a maximal sequence of states, i.e., a sequence such that there is no* σ' *such that* $\sigma_K \mapsto \sigma'$*. Then* $\sigma_K = \mathsf{valid}$ *or* $\sigma_K = \mathsf{invalid}$*.*

We prove the termination of the algorithm by defining a well-founded ordering on the tuples $\mathbf{R} \mid Q$.

Definition 1. *Let* $\mathbf{A}^{\downarrow} = (A_1{}^{\downarrow}, \ldots, A_N{}^{\downarrow})$ *and* $\mathbf{B}^{\downarrow} = (B_1{}^{\downarrow}, \ldots, B_N{}^{\downarrow})$ *be two finite sequences of downward-closed sets of the equal length* N*. Define* $\mathbf{A}^{\downarrow} \sqsubseteq \mathbf{B}^{\downarrow}$ *iff* $A_i{}^{\downarrow} \subseteq B_i{}^{\downarrow}$ *for all* $i = 1, \ldots, N$*. Let* Q *be a priority queue whose elements are tuples* $\langle a, i \rangle \in \Sigma \times \mathbb{N}$*, and let* N *be a natural number. Define* $\ell_N(Q) := \min(\{i \mid \langle a, i \rangle \in Q\} \cup \{N+1\})$*, to be the smallest priority in* Q *or* $N+1$ *if* Q *is empty.*

For two states $\mathbf{R} \mid Q$ *and* $\mathbf{R}' \mid Q'$*, such* $\text{length}(\mathbf{R}) = \text{length}(\mathbf{R}') = N$*, we define the ordering* $\leq_s$ *as:*

$$\mathbf{R} \mid Q \leq_s \mathbf{R}' \mid Q' :\Longleftrightarrow \mathbf{R} \sqsubseteq \mathbf{R}' \wedge (\mathbf{R} = \mathbf{R}' \rightarrow \ell_N(Q) \leq \ell_N(Q')$$

and we write $\mathbf{R} \mid Q <_s \mathbf{R}' \mid Q'$ *if* $\mathbf{R} \mid Q \leq_s \mathbf{R}' \mid Q'$ *but* $\mathbf{R} \neq \mathbf{R}'$ *or* $Q \neq Q'$*.*

Lemma 3 ($\leq_s$ is a well-founded quasi-order.). *The relation* $<_s$ *is a well-founded strict quasi-ordering on the set* $(\mathcal{D})^* \times \mathcal{Q}$*, where* $\mathcal{D}$ *is a set of downward-closed sets over* Σ*, and* $\mathcal{Q}$ *denotes the set of priority queues over* $\Sigma \times \mathbb{N}$*.*

The following proposition characterizes infinite runs of the algorithm. The proof follows from the observation that if $\mathbf{R} \mid Q \mapsto \mathbf{R}' \mid Q'$ as a result of applying the [Candidate], [Decide], [Conflict], or [Induction] rules, then $\mathbf{R} \mid Q >_s \mathbf{R}' \mid Q'$.

Proposition 2 (Infinite sequence condition). *For every infinite sequence* $\mathsf{Init} \mapsto \sigma_1 \mapsto \sigma_2 \mapsto \cdots$*, there are infinitely many* i *such that* $\sigma_i \mapsto \sigma_{i+1}$ *by applying the rule* [Unfold]*.*

We first prove that the algorithm terminates for the case when $\mathsf{Cover} \not\subseteq P^{\downarrow}$.

Lemma 4. *Let* $(\Sigma, I, \rightarrow, \preceq)$ *be a WSTS and* $P^{\downarrow}$ *a downward-closed set such that* $\mathsf{Cover}_k \cap (\Sigma \setminus P^{\downarrow}) \neq \emptyset$*. For every sequence* $\mathsf{Init} \mapsto \sigma_1 \mapsto^* \sigma_n$*, there are at most* k *different values for* i *such that* $\sigma_i \mapsto \sigma_{i+1}$ *using the* [Unfold] *rule.*

Theorem 3. *[Termination when* $\mathsf{Cover} \not\subseteq P^{\downarrow}$*] Given a WSTS* $(\Sigma, I, \rightarrow, \preceq)$ *and a downward-closed set* $P^{\downarrow}$*, if* $\mathsf{Cover} \not\subseteq P^{\downarrow}$*, then the algorithm terminates and all maximal execution sequences have the form* $\mathsf{Init} \mapsto^* \mathsf{invalid}$*.*

Proof. Since $\mathsf{Cover} \not\subseteq P^{\downarrow}$, there is a state $y \in \mathsf{Cover} \setminus P^{\downarrow}$. By the definition of Cover, there are states x', y' such that $x' \in I$, $y' \succeq y$ and $x' \rightarrow^k y'$ for some $k \geq 0$. Because $\Sigma \setminus P^{\downarrow}$ is upward-closed, we have $y' \in \Sigma \setminus P^{\downarrow}$. Combining Lemma 4 and Proposition 2, we prove that the algorithm terminates.

Let $\mathsf{Init} \mapsto^* \sigma$ be a maximal execution. By Proposition 1, $\sigma = \mathsf{valid}$ or $\sigma = \mathsf{invalid}$. By Theorem 1, $\sigma \neq \mathsf{valid}$. ☐

To prove that the algorithm terminates when $\mathsf{Cover} \subseteq P^{\downarrow}$, we use an additional assumption:

Apply [Valid] whenever it is applicable. (†)

This is natural assumption: since the algorithm is used to decide the coverability problem and [Valid] answers the problem positively, choosing the [Valid] rule when it is applicable is the most efficient choice. The main argument for showing the termination will reduce to showing that, for downward-finite WSTS, we can generate only a finite number of different sets $R_i^{\downarrow}$, so [Valid] will be applicable at some point. The key combinatorial property of downward-finite wqos is as follows.

Lemma 5. *Let $(\Sigma, \preceq)$ be a downward-finite wqo and let D be a finite set. Consider a sequence $R_0^{\downarrow} \subseteq R_1^{\downarrow} \subseteq \dots$, where each $R_i^{\downarrow} = \Sigma \setminus \{r_{i,1}, \dots, r_{i,m_i}\}\uparrow$ for $r_{i,j} \in D\downarrow$. Then there is $K \in \mathbb{N}$ such that $R_K = R_{K+1}$.*

Proof. By downward-finiteness, $D\downarrow$ is finite. Hence, there are only a finite number of different $R_i^{\downarrow}$s we can construct, and the sequence must converge. ◻

Theorem 4. *[Termination when $\mathsf{Cover} \subseteq P^{\downarrow}$] For a given downward-finite WSTS $(\Sigma, I, \to, \preceq)$ and a downward-closed set $P^{\downarrow}$, if $\mathsf{Cover} \subseteq P^{\downarrow}$ and the rule [Valid] is applied whenever possible, then the algorithm terminates and all maximal execution sequences have the form $\mathsf{Init} \mapsto^* \mathsf{valid}$.*

Proof. Consider any execution sequence $\mathsf{Init} \mapsto \sigma_1 \mapsto \sigma_2 \mapsto \cdots$. To show that it is finite, by Proposition 2, it is sufficient to show that there are only finitely many i such that $\sigma_i \mapsto \sigma_{i+1}$ via rule [Unfold]. Note that every time [Unfold] is applied, the length of the sequence $\mathbf{R}$ goes up. Consider the bound K obtained by applying Lemma 5 to the finite set $\bigcup_{i \geqslant 0} D_i$. After K applications of [Unfold], by Lemma 5, the [Valid] rule applies. Since [Valid] is taken whenever it is applied, the sequence must terminate. By soundness, it must terminate in valid. ◻

Note that Theorem 4 is the only result that requires downward-finiteness of the WSTS. We show that the downward-finiteness condition is necessary. Consider a WSTS $(\mathbb{N} \cup \{\omega\}, \{0\}, \to, \leqslant)$, where $x \to x+1$ for each $x \in \mathbb{N}$ and $\omega \to \omega$, and $\leqslant$ is the natural order on $\mathbb{N}$ extended with $x \leqslant \omega$ for all $x \in \mathbb{N}$. Consider the downward closed set $\mathbb{N}$. The backward analysis terminates in one step, since $\mathrm{pre}(\omega) = \{\omega\}$. However, the IC3 algorithm need not terminate. After unfolding, we find a conflict since $\mathrm{pre}(\omega) = \{\omega\}$, which is not initial. Generalizing, we get $R_1 = \{0, 1\}$. At this point, we unfold again. We find another conflict, and generalize to $R_2 = \{0, 1, 2\}$. We continue this way to generate an infinite sequence of steps without terminating.

4 Coverability for Petri Nets

Petri nets [11] are a widely used model for concurrent systems. They form a downward-finite class of WSTS. We now describe an implementation of our algorithm for the coverability problem for Petri nets.

4.1 Petri Nets

A Petri net (PN, for short) is a tuple (S, T, W), where S is a finite set of *places*, T is a finite set of *transitions* disjoint from S, and $W : (S \times T) \cup (T \times S) \to \mathbb{N}$ is the arc multiplicity function.

The semantics of a PN is given using *markings*. A marking is a function from S to $\mathbb{N}$. For a marking m and place $s \in S$, we say s has $m(s)$ tokens.

A transition $t \in T$ is *enabled* at marking m, written $m|t\rangle$, if $m(s) \geqslant W(s,t)$ for all $s \in S$. A transition t that is enabled at m can fire, yielding a new marking m' such that $m'(s) = m(s) - W(s,t) + W(t,s)$. We write $m|t\rangle m'$ to denote the transition from m to m' on firing t.

A PN (S, T, W) and an initial marking m_0 give rise to a WSTS $(\Sigma, \{m_0\}, \to, \preceq)$ as follows. The set of states Σ is the set of markings. There is a single initial state m_0. There is an edge $m \to m'$ if there is some transition $t \in T$ such that $m|t\rangle m'$. Finally, $m \preceq m'$ if for each $s \in S$, we have $m(s) \leqslant m'(s)$. It is easy to check that the compatibility condition holds: if $m_1|t\rangle m_2$ and $m_1 \preceq m_1'$, then there is a marking m_2' such that $m_1'|t\rangle m_2'$ and $m_2 \preceq m_2'$. Moreover, the wqo is downward-finite. The coverability problem for PNs is defined as the coverability problem on this WSTS.

We represent Petri nets as follows. Let $S = \{s_1, \ldots, s_n\}$ be the set of places. A marking m is represented as the tuple of natural numbers $(m(s_1), \ldots, m(s_n))$. A transition t is represented as a pair $(\mathbf{g}, \mathbf{d}) \in \mathbb{N}^n \times \mathbb{Z}^n$, where $\mathbf{g}$ represents the enabling condition, and $\mathbf{d}$ represents the difference between the number of tokens in a place if the transition fires, and the current number of tokens. Formally, $(\mathbf{g}, \mathbf{d})$ is defined as:

$$\begin{aligned}
\mathbf{g} &= (W(s_1,t), \ldots, W(s_n,t)) \\
\mathbf{d} &= (W(t,s_1) - W(s_1,t), \ldots, W(t,s_n) - W(s_n,t))\,.
\end{aligned}$$

We represent upward-closed sets with their minimal bases, which are finite sets of n-tuples of natural numbers. A downward-closed set is represented as its complement (which is an upward-closed set). The sets $R_i^{\downarrow}$, which are constructed during the algorithm run, are therefore represented as their complements. Such a representation comes naturally as the algorithm executes. Originally each set $R_i^{\downarrow}$ is initialized to contain all the states. The algorithm removes sets of states of the form $\mathbf{b}\!\uparrow$ from $R_i^{\downarrow}$, for some $\mathbf{b} \in \mathbb{N}^n$. If a set $\mathbf{b}\!\uparrow$ was removed from $R_i^{\downarrow}$, we say that states in $\mathbf{b}\!\uparrow$ are *blocked* by $\mathbf{b}$ at level i. At the end every $R_i^{\downarrow}$ becomes to a set of the form $\Sigma \setminus \{\mathbf{b}_1, \ldots, \mathbf{b}_l\}\!\uparrow$ and we conceptually represent $R_i^{\downarrow}$ with $\{\mathbf{b}_1, \ldots, \mathbf{b}_l\}$.

The implementation uses a succinct representation of $\mathbf{R}$, so called *delta-encoding* [8]. Let $R_i^{\downarrow} = \Sigma \setminus B_i\!\uparrow$ and $R_{i+1}^{\downarrow} = \Sigma \setminus B_{i+1}\!\uparrow$ for some finite sets B_i and B_{i+1}. Applying the invariant (I3) yields $B_{i+1} \subseteq B_i$. Therefore we only need to maintain a vector $\mathbf{F} = (F_0, \ldots, F_N, F_\infty)$ such that $\mathbf{b} \in F_i$ if i is the highest level where $\mathbf{b}$ was blocked. This is sufficient because $\mathbf{b}$ is also blocked on all lower levels. As an illustration, for $(R_0^{\downarrow}, R_1^{\downarrow}, R_2^{\downarrow}) = (\{\mathbf{i}_1, \mathbf{i}_2\}, \{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4\}, \{\mathbf{b}_2, \mathbf{b}_3\})$, the matching vector $\mathbf{F}$ might be $(F_0, F_1, F_2, F_\infty) = (\{\mathbf{i}_1, \mathbf{i}_2\}, \{\mathbf{b}_1, \mathbf{b}_4\}, \{\mathbf{b}_2, \mathbf{b}_3\}, \emptyset)$. The set F_∞ represents states that can never be reached.

4.2 Implementation Details and Optimizations

Our implementation follows the rules given in Figures 1 and 2. In addition, we use optimizations from [8]. The main difference between our implementation and [8] is in the interpretation of sets being blocked: in [8] those are cubes identified with partial assignments to boolean variables, whereas in our case those are upward-closed sets generated by a single state. Also, a straightforward adaptation of the implementation [8] would replace a SAT solver with a solver for integer difference logic, a fragment of linear integer arithmetic which allows the most natural encoding of Petri nets. However, we observed that Petri nets allow an easy and efficient way of computing predecessors and deciding relative inductiveness directly. Thus we were able to eliminate the overhead of calling the SMT solver.

Testing membership in $R_i^{\downarrow}$. Many of the rules given in Figures 1 and 2 depend on testing whether some state $\mathbf{a}$ is contained in a set $R_k^{\downarrow}$. Using the delta-encoded vector $\mathbf{F}$ this can be done by iterating over F_i for $k \leqslant i \leqslant N+1$ and checking if any of them contains a state $\mathbf{c}$ such that $\mathbf{c} \preceq \mathbf{a}$. If there is such a state, it blocks $\mathbf{a}$, otherwise $\mathbf{a} \in R_k^{\downarrow}$. If $k = 0$, we search for $\mathbf{c}$ only in F_0.

Implementation of the rules. The delta-encoded representation $\mathbf{F}$ also makes [Valid] easy to implement. Checking if $R_i^{\downarrow} = R_{i+1}^{\downarrow}$ reduces to checking if F_i is empty for some $i < N$. [Unfold] is applied when [Candidate] can no longer yield a bad state contained in $R_N^{\downarrow}$. It increases N and inserts an empty set to position N in the vector $\mathbf{F}$, thus pushing F_∞ from position N to $N+1$. We implemented rules [Initialize], [Candidate], [ModelSyn] and [ModelSem] in a straightforward manner.

Computing predecessors. In the rest of the rules we need to find predecessors $\mathrm{pre}(\mathbf{a}\!\uparrow)$ in $R_i^{\downarrow} \setminus \mathbf{a}\!\uparrow$, or conclude relative inductiveness if no such predecessors exist. The implementation in [8] achieves this by using a function *solveRelative()* which invokes the SAT solver. But *solveRelative()* also does two important improvements. In case the SAT solver finds a cube of predecessors, it applies *ternary simulation* to expand it further. If the SAT solver concludes relative inductiveness, it extracts information to conclude a generalized clause is inductive relative to some level $k \geqslant i$. We succeeded to achieve analogous effects in case of Petri nets by the following observations. While it is unclear what ternary simulation would correspond to for Petri nets, the following lemma shows how to compute the most general predecessor along a fixed transition directly.

Lemma 6. *Let $\mathbf{a} \in \mathbb{N}^n$ be a state and $t = (\mathbf{g}, \mathbf{d}) \in \mathbb{N}^n \times \mathbb{Z}^n$ be a transition. Then $\mathbf{b} \in \mathrm{pre}(\mathbf{a}\!\uparrow)$ is a predecessor along t if and only if $\mathbf{b} \succeq \max(\mathbf{a} - \mathbf{d}, \mathbf{g})$.*

Therefore, to find an element of $\mathrm{pre}(\mathbf{a}\!\uparrow)$ and $R_i^{\downarrow} \setminus \mathbf{a}\!\uparrow$, we iterate through all transitions $t = (\mathbf{g}, \mathbf{d})$ and find the one for which $\max(\mathbf{a} - \mathbf{d}, \mathbf{g}) \in R_i^{\downarrow} \setminus \mathbf{a}\!\uparrow$.

If there are no such transitions, then $\Sigma \setminus \mathbf{a}\!\uparrow$ is inductive relative to $R_i^{\downarrow}$. In that case, for each transition $t = (\mathbf{g}, \mathbf{d})$ the predecessor $\max(\mathbf{a}-\mathbf{d}, \mathbf{g})$ is either blocked by $\mathbf{a}$ itself, or there is $i_t \geqslant i$ and a state $\mathbf{c}_t \in F_{i_t}$ such that $\mathbf{c}_t \preceq \max(\mathbf{a} - \mathbf{d}, \mathbf{g})$. We define

$$i' := \min\{i_t \mid t \text{ is a transition}\}\,,$$

where $i_t := N+1$ for $t = (\mathbf{g}, \mathbf{d})$ if $\max(\mathbf{a} - \mathbf{d}, \mathbf{g})$ is blocked by $\mathbf{a}$ itself. Then $i' \geqslant i$ and $\Sigma \setminus \mathbf{a}\!\uparrow$ is inductive relative to $R^{\downarrow}_{i'}$.

Computing generalizations. The following lemma shows that we can also significantly generalize $\mathbf{a}$, i.e. there is a simple way to compute a state $\mathbf{a}' \preceq \mathbf{a}$ such that for all transitions $t = (\mathbf{d}, \mathbf{g})$, $\max(\mathbf{a}' - \mathbf{d}, \mathbf{g})$ remains blocked either by $\mathbf{a}'$ itself, or by $\mathbf{c}_t$.

Lemma 7. *Let $\mathbf{a}, \mathbf{c} \in \mathbb{N}^n$ be states and $t = (\mathbf{g}, \mathbf{d}) \in \mathbb{N}^n \times \mathbb{Z}^n$ be a transition.*

1. *Let $\mathbf{c} \preceq \max(\mathbf{a} - \mathbf{d}, \mathbf{g})$. Define $\mathbf{a}'' \in \mathbb{N}^n$ by $a''_j := c_j + d_j$ if $g_j < c_j$ and $a''_j := 0$ if $g_j \geqslant c_j$, for $j = 1, \ldots, n$. Then $\mathbf{a}'' \preceq \mathbf{a}$. Additionally, for each $\mathbf{a}' \in \mathbb{N}^n$ such that $\mathbf{a}'' \preceq \mathbf{a}' \preceq \mathbf{a}$, we have $\mathbf{c} \preceq \max(\mathbf{a}' - \mathbf{d}, \mathbf{g})$.*
2. *If $\mathbf{a} \preceq \max(\mathbf{a} - \mathbf{d}, \mathbf{g})$, then for each $\mathbf{a}' \in \mathbb{N}^n$ such that $\mathbf{a}' \preceq \mathbf{a}$, it holds that $\mathbf{a}' \preceq \max(\mathbf{a}' - \mathbf{d}, \mathbf{g})$.*

To continue with the case when the predecessor $\max(\mathbf{a} - \mathbf{d}, \mathbf{g})$ is blocked for each transition $t = (\mathbf{d}, \mathbf{g})$, we define $\mathbf{a}''_t$ as in Lemma 7 (1) if the predecessor is blocked by some state $\mathbf{c}_t \in F_{i_t}$ and $\mathbf{a}''_t := (0, \ldots, 0)$ if it is blocked by $\mathbf{a}$ itself. The state $\mathbf{a}''$ is defined to be the pointwise maximum of all states $\mathbf{a}''_t$. By Lemma 7, predecessors of $\mathbf{a}''$ remain blocked by the same states $\mathbf{c}_t$ or by $\mathbf{a}''$ itself.

However, $\mathbf{a}''$ still does not have to be a valid generalization, because it might be in $R^{\downarrow}_0$. If that is the case, we take any state $\mathbf{c} \in F_0$ which blocks $\mathbf{a}$ (such a state exists because $\mathbf{a} \notin R^{\downarrow}_0$). Then $\mathbf{a}' := \max(\mathbf{a}'', \mathbf{c})$ is a valid generalization: $\mathbf{a}' \preceq \mathbf{a}$ and $\Sigma \setminus \mathbf{a}'\!\uparrow$ is inductive relative to $R^{\downarrow}_{i'}$.

Using this technique, rules [Decide], [Conflict] and [Induction] become easy to implement. Note that some additional handling is needed in rules [Conflict] and [Induction] when blocking a generalized upward-closed set $\mathbf{a}'\!\uparrow$. If $\Sigma \setminus \mathbf{a}'\!\uparrow$ is inductive relative to $R^{\downarrow}_{i'}$ for $i' < N$, we update the vector $\mathbf{F}$ by adding $\mathbf{a}'$ to $F_{i'+1}$. However, if $i' = N$ or $i' = N+1$, we add $\mathbf{a}'$ to $F_{i'}$. Additionaly, for $1 \leqslant k \leqslant i'+1$ (or $1 \leqslant k \leqslant i'$) we remove all states $\mathbf{c} \in F_k$ such that $\mathbf{a}' \preceq \mathbf{c}$.

One of the optimizations from [8] showed a significant improvement in running time. After using the [Conflict] rule, if $i'+1 < N$ and a set $\mathbf{a}\!\uparrow$ was blocked from $R^{\downarrow}_{i'+1}$ by adding a generalization $\mathbf{a}'$ to $F_{i'+1}$, we add $\langle \mathbf{a}, i'+2 \rangle$ to the priority queue. This way we do not discard the state which we know leads outside $P^{\downarrow}$, but add an obligation to check if its upward-closure can be reached in $i'+2$ steps. The effect is that traces much longer than N are checked.

5 Experimental Evaluation

We have implemented the IC3 algorithm in a tool called IIC. Our tool is written in C++ and uses the input format of mist2. We evaluated the efficiency of the algorithm on a collection of Petri net examples. The goal of the evaluation was to compare the performance —both time and space usage— of IIC against other implementations of Petri net coverability.

Problem Instance	IIC Time	IIC Mem	Backward Time	Backward Mem	EEC Time	EEC Mem	MCOV Time	MCOV Mem
Uncoverable instances								
Bingham ($h = 150$)	**0.1**	**3.5**	970.3	146.3	1.8	19.0	**0.1**	7.6^{2c}
Bingham ($h = 250$)	**0.2**	**6.7**	Timeout		9.6	45.4	**0.2**	19.6^{2c}
Ext. ReadWrite (small consts)	**0.0**	**1.3**	0.1	3.7	Timeout		Timeout/OOM	
Ext. ReadWrite	**0.3**	**1.5**	216.3	34.1	Timeout		0.6	4.1^{2b}
FMS (old)	< **0.1**	**1.3**	1.3	5.5	Timeout		0.1	5.8^{2c}
Mesh2x2	< **0.1**	**1.3**	0.3	3.9	266.9	24.3	< **0.1**	4.2^{1c}
Mesh3x2	< **0.1**	**1.5**	4.1	7.0	Timeout		< **0.1**	2.0^{2b}
Multipoll	1.5	**1.6**	0.5	4.3	21.8	7.1	< **0.1**	1.7^{2b}
MedAA1	**0.5**	**173.3**	8.8	598.8			3.7	210.4^{2b}
MedAA2	Timeout		Timeout				Timeout/OOM	
MedAA5	Timeout		Timeout				Timeout/OOM	
MedAR1	**0.8**	**173.3**	8.77	598.8			3.7	210.4^{2b}
MedAR2	33.2	**173.3**	15.7	599.4			**13.7**	210.4^{2b}
MedAR5	128.1	**173.3**	26.6	600			**12.9**	210.4^{2b}
MedHA1	**0.8**	**173.3**	8.9	598.7			5.5^{2c}	210.4^{2b}
MedHA2	33.2	**173.3**	14.7	599.5			**12.6**	210.4^{2b}
MedHA5	Timeout		3219.7	647.3			12.5	210.4^{2b}
MedHQ1	**0.7**	**173.3**	8.8	598.8			12.2	210.4^{2b}
MedHQ2	33.8	**173.3**	16.6	596.9			**13.2**	210.4^{2b}
MedHQ5	125.8	**173.3**	26.6	600			**12.6**	210.4^{2b}
Coverable instances								
Kanban	< **0.1**	**1.4**	804.7	55.1	Timeout		0.1	6.0^{2c}
pncsacover	2.8	**2.2**	7.9	11.2	36.5	8.8	**1.0**	23.0^{1c}
pncsasemiliv	0.1	**1.5**	0.2	3.9	32.1	8.8	< **0.1**	3.7^{2c}
MedAA1-bug	**0.8**	**172.7**	1.0	596.9	56.5	658.0	3.6	210.4^{2b}
MedHR2-bug	**0.6**	**172.7**	0.6	596.9	57.2	658.0	12.8	210.4^{2b}
MedHQ2-bug	0.4	**172.7**	**0.3**	596.9	56.8	658.0	12.9	210.4^{2b}

Table 1. Experimental results: comparison of running time and memory consumption for different coverability algorithms on selected problem instances. The memory consumption is given in megabytes, and the running time in seconds. In the mcov column, the superscripts indicate the version of bfc used (^{1} means the version Jan 2012 version, ^{2} the Feb 2013 version), and the analysis mode (^{c}: combined, ^{b}: backward only, ^{f}: forward only). We list the best result for all the version/parameter combinations that were tried.

We compare the performance of IIC, using our implementation described above, to the following algorithms: EEC [13] and backward search [1], as implemented by the tool mist2[1], and the MCOV algorithm [16] for parameterized multithreaded programs as implemented by bfc[2]. All experiments were performed on identical machines, each having Intel Xeon 2.67 GHz CPUs and 48 GB of memory, running Linux 3.2.21 in 64 bit mode. Execution time was limited to 1 hours, and memory to five gigabytes.

We used 29 Petri net examples from the mist2 distribution, 46 examples of multi-threaded programs from the bfc distribution, and 6 examples from checking security properties of message-passing programs communicating through unbounded unordered channels (MedXXX examples). We only present a selection of the data and focus on examples that took longer than 2 second for at least one

[1] See `http://software.imdea.org/~pierreganty/ist.html`

[2] See `http://www.cprover.org/bfc/`

Problem Instance	IIC Time	IIC Mem	MCOV Time	MCOV Mem
Uncoverable instances				
Conditionals 2	0.1	**3.6**	< **0.1**	5.7^{2c}
RandCAS 2	< **0.1**	**2.0**	< **0.1**	3.9^{2c}
Coverable instances				
Boop 2	82.0	287.9	**0.1**	$\mathbf{12.1}^{1c}$
FuncPtr3 1	< 0.1	**1.5**	< 0.1	3.4^{2c}
FuncPtr3 2	0.2	**12.3**	0.1	7.9^{2c}
FuncPtr3 3	28.5	939.1	**3.6**	$\mathbf{303.8}^{1c}$
DoubleLock1 2	Timeout		**0.8**	$\mathbf{56.7}^{2c}$
DoubleLock3 2	8.0	41.3	< **0.1**	$\mathbf{4.8}^{2c}$
Lu-fig2 3	Timeout		**0.1**	$\mathbf{10.4}^{2c}$
Peterson 2	Timeout		**0.2**	$\mathbf{23.0}^{1c}$
Pthread5 3	132428	468.8	**0.1**	$\mathbf{17.0}^{1c}$
Pthread5 3			0.2	$\mathbf{49.6}^{2c}$
SimpleLoop 2	7.9	6.0	< **0.1**	$\mathbf{4.8}^{2c}$
Spin2003 2	4852.2	54.4	< **0.1**	$\mathbf{2.7}^{2c}$
StackCAS 2	2.5	**1.6**	< **0.1**	3.7^{2c}
StackCAS 3	5.5	21.7	< **0.1**	$\mathbf{4.4}^{2c}$
Szymanski 2	Timeout		**0.4**	$\mathbf{26.7}^{2c}$

Table 2. Experimental results: comparison between MCOV and IIC on examples derived from parameterized multithreaded programs. In the mcov column, the superscripts indicate the version of bfc used (^{1} means the version Jan 2012 version, ^{2} the Feb 2013 version), and the analysis mode (^{c}: combined, ^{b}: backward only, ^{f}: forward only). We list the best result for all the version/parameter combinations that were tried.

algorithm. All benchmarks are available at `http://www.mpi-sws.org/~jkloos/iic-experiments`.

mist2 and MedXXX benchmarks Table 1 show run times and memory usage on the mist2 and message-passing program benchmarks. For each row, the column in bold shows the winner (time or space) for each instance. It can be seen that IIC performs reasonably well on these benchmarks, both in time and in memory usage.

To account for mist2's use of a pooled memory, we estimated its baseline usage to 2.5 MB by averaging over all examples that ran in less than 1 second.

Multithreaded program benchmarks We also ran comparisons with MCOV on a set of multithreaded programs distributed with MCOV. For Petri nets derived from C programs distributed with MCOV, Table 2 shows that IIC performs well on the uncoverable examples but MCOV performs much better on the coverable ones. We do not fully understand the reasons for poor performance of IIC for the coverable instances.

In conclusion, the unoptimized implementation of the IIC algorithm is already working quite well in comparison to other existing implementations of coverability algorithms. Nevertheless, it is obvious that significant further work is required to optimize the algorithm. Two main directions that are being considered are the use of invariants to prune the search space, and the combination of the generalization heuristics from MCOV [16] with IIC.

Acknowledgements We thank Andreas Kaiser for pointing out an error regarding the encoding of Petri nets into the bfc input format, leading to non-optimal performance of the bfc tool, and for providing us with a correct conversion tool.

References

1. P. A. Abdulla, K. Cerans, B. Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *LICS '96*, pages 313–321. IEEE, 1996.
2. P.A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In *CAV'98*, LNCS 1427, pages 305–318. Springer, 1998.
3. A.R. Bradley. SAT-based model checking without unrolling. In *VMCAI'11*, LNCS, pages 70–87. Springer, 2011.
4. G. Ciardo. Petri nets with marking-dependent arc multiplicity: properties and analysis. In *ICATPN '94*, volume 815 of *LNCS*, pages 179–198. Springer, 1994.
5. A. Cimatti and A. Griggio. Software model checking via IC3. In *CAV'12: Computer-Aided Verification*, LNCS 7358, pages 277–293. Springer, 2012.
6. L.E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics*, 35(4):413–422, 1913.
7. C. Dufourd, A. Finkel, and P. Schnoebelen. Reset nets between decidability and undecidability. In *ICALP '98*, LNCS 1443, pages 103–115. Springer, 1998.
8. N. Een, A. Mishchenko, and R. Brayton. Efficient implementation of property directed reachability. In *FMCAD'11*, pages 125–134. FMCAD Inc, 2011.
9. E.A. Emerson and K.S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *LICS '98*, pages 70–80. IEEE, 1998.
10. J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *LICS '99*, pages 352–359. IEEE Computer Society, 1999.
11. J. Esparza and M. Nielsen. Decidability issues for petri nets - a survey. *Bulletin of the EATCS*, 52:244–262, 1994.
12. A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.
13. G. Geeraerts, J.-F. Raskin, and L. Van Begin. Expand, enlarge and check: New algorithms for the coverability problem of WSTS. *J. Comput. Syst. Sci.*, 72(1):180–203, February 2006.
14. G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, s3-2(1):326–336, 1952.
15. K. Hoder and N. Bjørner. Generalized property directed reachability. In *SAT'12*, pages 157–171. Springer, 2012.
16. A. Kaiser, D. Kroening, and T. Wahl. Efficient coverability analysis by proof minimization. *CONCUR 2012–Concurrency Theory*, pages 500–515, 2012.

A Soundness and termination proof

This appendix contains the proofs of lemmas used in the paper.

Lemma 1 *The rules* [Unfold]*,* [Induction]*,* [Conflict]*,* [CandidateNondet]*, and* [DecideNondet] *preserve* (I1) – (I4)*,*

Proof. If $\mathbf{R}|Q \mapsto \mathbf{R}'|Q'$ by application of [CandidateNondet] or [DecideNondet], we have $\mathbf{R} = \mathbf{R}'$, so (I1) – (I4) are preserved trivially. For [Unfold], (I1) – (I3) are trivial, and (I4) holds for $i < \text{length}\,\mathbf{R}' - 1$ by (I4) on $\mathbf{R}$, and by the condition of [Unfold] for $i = \text{length}\,\mathbf{R}' - 1$.

Finally, the rules [Induction] and [Conflict] require the following technical observation about Gen.

> **Claim:** If $\mathbf{R}$ satisfies (I1) – (I4) and $b \in \text{Gen}_m(a)$, then $\mathbf{R}[R_i^{\downarrow} \leftarrow R_i^{\downarrow} \setminus b\!\uparrow]_{i=1}^m$ satisfies (I1) – (I4).

To prove the claims, we show the following:

1. For $1 \leq k \leq i$, $I \subseteq R_k^{\downarrow} \setminus b\!\uparrow$ (part of (I1)).
2. $\text{post}(R_{i-1}^{\downarrow} \setminus b\!\uparrow) \subseteq R_i^{\downarrow} \setminus a'\!\uparrow$. for $1 < i < m$ (Part of (I2)).
3. $\text{post}(R_0^{\downarrow}) \subseteq R_1^{\downarrow} \setminus b\!\uparrow$. ((I2), case $i = 1$)
4. $\text{post}(R_{m-1}^{\downarrow} \setminus b\!\uparrow) \subseteq R_m^{\downarrow}$ ((I2), case $i = m$)

All other cases as well as (I3) and (I4) are trivial.

1. By the definition of Gen, we have $b\!\uparrow \cap I = \varnothing$. Thus, since $I \subseteq R_i^{\downarrow}$ by (I1), $I \subseteq R_i^{\downarrow} \setminus b\!\uparrow$.
2. Let i be given with $1 < i < m$, and $y \in \text{post}(R_{i-1}^{\downarrow} \setminus b\!\uparrow)$. We need to show that $y \in R_i^{\downarrow} \setminus b\!\uparrow$.
   By choice of y, there is an $x \in R_{i-1}^{\downarrow} \setminus b\!\uparrow$ such that $x \to y$. By repeated application of (I3), we find that $x \in R_{m-1}^{\downarrow} \setminus b\!\uparrow$. Thus, $y \in \text{post}(R_{m-1}^{\downarrow} \setminus b\!\uparrow) \subseteq \Sigma \setminus b\!\uparrow$.
   Thus, $y \in R_i^{\downarrow} \setminus b\!\uparrow$.
3. Let $y' \in \text{post}(R_0^{\downarrow}) = \text{post}(I\!\downarrow)$. We need to show that $y' \in R_1^{\downarrow} \setminus b\!\uparrow$.
   There is a $x' \in I\!\downarrow$ such that $x' \to y'$. Due to the choice of x', there is an $x \in I$ with $x \succeq x'$. By well-structuredness, there is also a y such that $x \to y$ and $y \succeq y'$. Since $R_1^{\downarrow}$ is downward-closed, $y \in R_1^{\downarrow}$.
   By (I3), we find that $x \in R_1^{\downarrow}$, and by (1), $x \in R_1^{\downarrow} \setminus b\!\uparrow$. Thus, by (2), $y \in R_2^{\downarrow} \setminus b\!\uparrow$. But this implies $y \notin b\!\uparrow$, so $y \in R_1^{\downarrow} \setminus b\!\uparrow$. Since $R_1^{\downarrow} \setminus b\!\uparrow$ is downward-closed, we hence have $y' \in R_1^{\downarrow} \setminus b\!\uparrow$.
4. $\text{post}(R_{m-1}^{\downarrow} \setminus b\!\uparrow) \subseteq \text{post}(R_{m-1}^{\downarrow}) \subseteq R_m^{\downarrow}$ by (I2).

□

The next lemma defines the structure of the priority queues used in the algorithm.

Lemma 2 *Let* $\mathsf{Init} \mapsto^* \mathbf{R} \mid Q$*. If* $Q \neq \varnothing$*, then for every* $\langle a, i \rangle \in Q$*, there is a path from* a *to some* $b \in \Sigma \setminus P^{\downarrow}$*.*

Proof. By induction on the application of rules. For the base case, the application of [$\mathsf{Initialize}$], the claim trivially holds.

For the induction step, assume the claim holds for some sequence of rule applications such that $\mathsf{Init} \mapsto^* \mathbf{R} \mid Q$. We only need to consider [$\mathsf{CandidateNondet}$] and [$\mathsf{DecideNondet}$], since they are the only rules which add elements on Q.

If [$\mathsf{CandidateNondet}$] is applied, it will enqueue $\langle a, N \rangle$ such that $a \in R_N^{\downarrow} \setminus P^{\downarrow} \subseteq \Sigma \setminus P^{\downarrow}$. If [$\mathsf{DecideNondet}$] is applied, then $\min Q = \langle a, i \rangle$, $i > 0$ and $\langle b', i-1 \rangle$ such that $b' \in \mathrm{pre}(a\!\uparrow)$ is enqueued. The latter implies there is $a' \succeq a$ such that $b' \to a'$. By the induction hypothesis, there is a path $a \to^* b \in \Sigma \setminus P^{\downarrow}$, therefore by well-structuredness there is $b'' \succeq b$ such that $a' \to^* b''$. Combining the facts we conclude $b' \to^* b'' \in \Sigma \setminus P^{\downarrow}$. □

Lemma 8 (Disjointness of $R_i^{\downarrow}$ and U_j). *When* (I1) – (I4) *hold for* $\mathbf{R}$*,* $R_{N-1-i} \cap U_i = \varnothing$ *for* $0 \leq i < N-1$*.*

Proof. We prove the statement by induction over i.

$i = 0$: By (I4),

$$R_{N-1-0} \cap U_0 = \underbrace{R_{N-1}}_{\subseteq P^{\downarrow}} \cap (\Sigma \setminus P^{\downarrow}) = \varnothing.$$

$i > 0$: By induction, $R_{N-i} \cap U_{i-1} = \varnothing$. Now, let $x \in U_i$. Then by definition of U_i, there are two cases:

$x \in U_{i-1}$: Then $x \notin R_{N-i}^{\downarrow}$. Since $R_{N-i-1}^{\downarrow} \subseteq R_{N-i}$ by (I3), $x \notin R_{N-i-1}^{\downarrow}$.

$x \in \mathrm{pre}(U_{i-1})$: Then there is a $y \in U_{i-1}$ such that $x \to y$. In particular, $y \in \mathrm{post}(x)$. Since $y \in U_{i-1}$, we also have $y \notin R_{N-i}^{\downarrow}$. By (I2) and $z \in R_{N-i-1}^{\downarrow} \Rightarrow \mathrm{post}(z) \subseteq \mathrm{post}(R_{N-i-1}^{\downarrow})$, this implies $x \notin R_{N-i-1}^{\downarrow}$.

Thus, in either case, $x \notin R_{N-i-1}^{\downarrow}$. This implies $R_{N-i-1}^{\downarrow} \cap U_i = \varnothing$.

□

Lemma 9. *The sets* D_i *satisfy the following properties:*

1. $D_0 \subseteq \Sigma \setminus P^{\downarrow}$
2. $D_{i+1} \subseteq \mathrm{pre}(D_i) \setminus U_i$
3. *Whenever* $R_N^{\downarrow} \setminus P^{\downarrow} \neq \varnothing$*, there exists an* $x \in R_N^{\downarrow} \cap D_0$
4. *For all* $a \in D_i$*, if* $\mathrm{pre}(a\!\uparrow) \cap R_{N-i-1}^{\downarrow} \neq \varnothing$*, there exists an element* x *such that* $x \in \mathrm{pre}(a\!\uparrow) \cap D_{i+1} \cap R_{N-i-1}^{\downarrow}$
5. D_i *is finite for all* $i \geq 0$

Proof. Statements 1) and 2) follow trivially.

To prove (3), assume that $y \in R_N^{\downarrow} \setminus P^{\downarrow}$. Then there is a minimal element $x \in \min(R_N^{\downarrow} \setminus P^{\downarrow})$. But since $R_N^{\downarrow}$ is downward-closed, $\min(R_N^{\downarrow} \setminus P^{\downarrow}) \subseteq \min(\Sigma \setminus P^{\downarrow})$. Thus, $x \in \min(\Sigma \setminus P^{\downarrow}) = D_0$. $x \in R_N^{\downarrow}$ is clear.

To show (4), let $a \in D_i$ be given, and assume that $y \in \mathrm{pre}(a\uparrow) \cap R^{\downarrow}_{N-i-1}$. Again, there is a minimal element $x \in \mathrm{pre}(a\uparrow) \cap R^{\downarrow}_{N-i-1}$. By Lemma 8, $x \notin U_i$. Thus, $x \in D_{i+1}$.

Finally, (5) follows by induction on i: For $i = 0$, the statement is clear because of the finiteness of min. For $i > 0$, the set D_{i-1} is finite by induction hypothesis. Thus, the union $\bigcup_{a \in D_{i-1}} \min(a\uparrow)$ is a finite union over finite sets, thus D_i is a subset of a finite set and hence finite. ◻

Lemma 10. *Given a WSTS $(\Sigma, I, \to, \preceq)$, a downward-closed set $P^{\downarrow}$ and a sequence of sets D_i, if* $\mathsf{Init} \mapsto^* \mathbf{R}|Q$*, then:*

1. *For all $i \geq 1$, $R^{\downarrow}_i = \Sigma \setminus \{r_{i,1}, \dots, r_{i,m_i}\}$, where for all $j = 1, \dots, m_i$, there is a $k \geq 0$ and a $d \in D_k$ such that $r_{i,j} \leq d$.*
2. *For all $\langle a, i \rangle \in Q$, $a \in D_{N-i}$.*

Proof. It is again sufficient to show that $I\downarrow|\varnothing$ has this property, and that all relevant rules preserve it. Since $I\downarrow|\varnothing$ satisfies the requirements vacuously, assume that $\mathbf{R}|Q \mapsto \mathbf{R}'|Q'$. By inspection, the following five rules need to be considered:

[Unfold] Trivial.

[Induction] Since $Q = Q'$, the second part is trivial.

For the first part, let $b \in \mathrm{Gen}_i(r_{i,j})$ for given i, j. By the definition of Gen, $b \leq r_{i,j}$, and by induction hypothesis, $r_{i,j} \leq d$ for some $d \in D_k$, $k > 0$. By transitivity, $b \leq d$. Furthermore,

$$R'^{\downarrow}_{\ell} = \begin{cases} \Sigma \setminus \{r_{\ell,1}, \dots, r_{\ell,m_\ell}\}\uparrow \setminus b\uparrow & 1 \leq \ell \leq i+1 \\ \Sigma \setminus \{r_{\ell,1}, \dots, r_{\ell,m_\ell}\}\uparrow & \text{otherwise} \end{cases}$$
$$= \begin{cases} \Sigma \setminus \{r_{\ell,1}, \dots, r_{\ell,m_\ell}, b\}\uparrow & 1 \leq \ell \leq i+1 \\ \Sigma \setminus \{r_{\ell,1}, \dots, r_{\ell,m_\ell}\}\uparrow & \text{otherwise} \end{cases}$$

So, in either case, $R'^{\downarrow}_{\ell}$ is of the required form.

[Candidate] Trivial.

[Decide] Trivial.

[Conflict] Since $Q' \subseteq Q$, the second part is trivial. For the first part, we have $b \in \mathrm{Gen}_i(a)$ for some $a \in D_k$, $k \geq 0$. Thus, $b \leq a$ by the definition of Gen. The rest of the proof is analogous to the case of [Induction].

◻

Lemma 11 (Progress). *If* $\mathsf{Init} \mapsto^* \mathbf{R}|Q$*, then either* $\mathbf{R}|Q \mapsto \mathbf{R}'|Q'$*, or* $\mathbf{R}|Q \mapsto$ *valid, or* $\mathbf{R}|Q \mapsto$ *invalid.*

Proof. Let $\mathbf{R}|Q$ be given. By case analysis, we will show that some rule will always be applicable to it.

If $Q = \varnothing$, there are two cases:

– $R^{\downarrow}_N \subseteq P^{\downarrow}$.
  $\mathbf{R}|Q = \mathbf{R}|\varnothing \mapsto \mathbf{R} \cdot \Sigma|\varnothing$ by applying [Unfold].

- $R_N^{\downarrow} \not\subseteq P^{\downarrow}$.
  Then, by choice of D_0, there is some $x \in R_N^{\downarrow} \cap D_0$. Thus, $\mathbf{R}|Q = \mathbf{R}|\varnothing \mapsto \mathbf{R}|\langle x, N\rangle$ by applying [$\mathsf{Candidate}$].

If $Q \neq \varnothing$ is not empty, there are four cases:

- $\langle a, 0\rangle \in Q$ for some $a \in \Sigma$.
  $\mathbf{R}|Q \mapsto \mathsf{invalid}$ by applying [$\mathsf{ModelSyn}$].
- $\langle a, i\rangle \in Q$ for some $a \in \Sigma$, $i \geq 0$ with $a\!\uparrow\!\cap I \neq \varnothing$.
  $\mathbf{R}|Q \mapsto \mathsf{invalid}$ by applying [$\mathsf{ModelSem}$].
- $\min Q = \langle a, i\rangle$ for some $a \in \Sigma$, $i > 0$ with $\mathrm{pre}(a\!\uparrow) \cap R_{i-1}^{\downarrow} \neq \varnothing$.
  By choice of D_{N-i+1}, there is also a $b \in D_{N-i+1} \cap R_{i-1}^{\downarrow} \cap \mathrm{pre}(a\!\uparrow)$, so $\mathbf{R}|Q \mapsto \mathbf{R}|Q.\textsc{Push}(\langle b, i-1\rangle)$ by applying [Decide].
- None of the above.
  In this case, let $\langle a, i\rangle = \min Q$. We have $i > 0$, $a\!\uparrow\!\cap I = \varnothing$ and $\mathrm{pre}(a\!\uparrow) \cap R_{i-1}^{\downarrow} = \varnothing$.
  Claim: $a \in \mathrm{Gen}_i(a)$.
  Proof: We certainly have $a \leq a$, and by the statements above, $a\!\uparrow\!\cap I = \varnothing$.
    Also, by Lemma 2, $a \in R_i^{\downarrow}$. It remains to show that $\mathrm{post}(R_{i-1}^{\downarrow} \setminus a\!\uparrow) \subseteq \Sigma \setminus a\!\uparrow$.
    Thus, let $y \in \mathrm{post}(R_{i-1}^{\downarrow} \setminus a\!\uparrow)$. Then there is an $x \in R_{i-1}^{\downarrow} \setminus a\!\uparrow$ such that $x \to y$.
    Suppose now that $y \in a\!\uparrow$. Then $x \in \mathrm{pre}(a\!\uparrow)$, so $x \in \mathrm{pre}(a\!\uparrow) \cap R_{i-1}^{\downarrow} = \varnothing$ – contradiction.
    Thus, $a \in \mathrm{Gen}_i(a)$.

  Thus, $\mathbf{R}|Q \mapsto (\mathbf{R}[R_k^{\downarrow} \leftarrow R_k^{\downarrow} \setminus a\!\uparrow]_{k=1}^{i})|(Q.\textsc{PopMin})$ by applying [$\mathsf{Conflict}$].

□

Proposition 1 *[Maximal finite sequences] Let $\mathsf{Init} = \sigma_0 \mapsto \sigma_1 \mapsto \cdots \mapsto \sigma_K$ be a maximal sequence of states, i.e., a sequence such that there is no σ' such that $\sigma_K \mapsto \sigma'$. Then $\sigma_K = \mathsf{valid}$ or $\sigma_K = \mathsf{invalid}$.*

Proof. σ_K can have four values, Init, valid, $\mathsf{invalid}$ or $\mathbf{R}|Q$.
If $\sigma_K = \mathsf{valid}$ or $\sigma_K = \mathsf{invalid}$.
If $\sigma_K = \mathbf{R}|Q$, the sequence is not maximal by Lemma 11.
If $\sigma_K = \mathsf{Init}$, $\sigma_K \mapsto I\!\downarrow\!|\varnothing$, by [$\mathsf{Initialize}$], hence the sequence is not maximal.

□

Lemma 3 *[$\leq_s$ is a well-founded quasi-order.] The relation $<_s$ is a well-founded strict quasi-ordering on the set $(\mathcal{D})^* \times \mathcal{Q}$, where $\mathcal{D}$ is a set of downward-closed sets over Σ, and $\mathcal{Q}$ denotes the set of priority queues over $\Sigma \times \mathbb{N}$.*

Proof. The following statements are easy to check:

- $\sqsubseteq_N$ is a partial order, and $\sqsubset_N$ is its strict part.
- $\sqsubseteq$ is a partial order, and $\sqsubset$ is its strict part.

– Let $\leq_n := \sqsubseteq \times_{\text{lex}} \leq$ denote the lexicographical product of $\sqsubseteq$ and the order $\leq$ on the natural numbers. Then $\leq_n$ is a partial order.
– Let $\phi : (\mathcal{D})^* \times \mathcal{Q} \to (\mathcal{D})^* \times \mathbb{N}, \mathbf{R}|Q \mapsto (\mathbf{R}, \ell_{\text{length}(\mathbf{R})}(Q))$.
Then $\phi(\mathbf{R}|Q) <_n \phi(\mathbf{R}'|Q')$ if $\mathbf{R}|Q <_s \mathbf{R}'|Q'$, and $\phi(\mathbf{R}|Q) \leq_n \phi(\mathbf{R}'|Q')$ if $\mathbf{R}|Q \leq_s \mathbf{R}'|Q'$.
– If $\leq_s$ is a quasi-order, $<_s$ is the corresponding strict quasi-order.

In the following, we will use these facts to establish:

1. $\sqsubseteq$ is well-founded,
2. $\leq_n$ is well-founded,
3. $\leq_s$ is a quasi-order,
4. $\leq_s$ is well-founded.

$\sqsubseteq$ **is well-founded:** Let $\mathbf{R}_1 \sqsupseteq \mathbf{R}_2 \sqsupseteq \cdots$ be a descending chain of vectors. We need to show that the chain will eventually stabilize, i.e., there is an i such that for all $j \geq i$, $\mathbf{R}_j = \mathbf{R}_i$.
As a first observation, by definition of $\sqsubseteq$, length $\mathbf{R}_j$ = length $\mathbf{R}_{j+1}$ for all $j \geq 0$, i.e., there is an N such that length $\mathbf{R}_j = N$ for all j.
Suppose that no such i exists. Then for all j, $\mathbf{R}_{j+1} \sqsupset_N \mathbf{R}_j$. By definition of $\sqsupset_N$, this means that for every j, there is a k_j such that $R^{\downarrow}_{j,k_j} \supsetneq R^{\downarrow}_{j+1,k_j}$. Furthermore, since $k_j \in \{1, \ldots N\}$ for all j, there must be some $k \in \{1, \ldots, N\}$ such that $k_j = k$ for infinitely many j by the pigeonhole principle.
Define a sequence j_t such that $j_0 = 0$, and for all $t \geq 0$, $R^{\downarrow}_{j_t,k} = R^{\downarrow}_{j_{t+1}-1,k} \supsetneq R^{\downarrow}_{j_t,k}$. Such a sequence exists because for every j, either $R^{\downarrow}_{j,k} = R^{\downarrow}_{j+1,k}$, or $R^{\downarrow}_{j,k} \supsetneq R^{\downarrow}_{j+1,k}$ by the assumptions.
Thus, we have an infinite descending chain $R^{\downarrow}_{j_0,k} \supsetneq R^{\downarrow}_{j_1,k} \supsetneq \ldots$ of downward-closed sets. Define $C^{\uparrow}_t := \Sigma \setminus R^{\downarrow}_{j_t,k}$. This is an infinite strictly ascending chain of upward-closed sets, i.e., $C^{\uparrow}_0 \subsetneq C^{\uparrow}_1 \subsetneq \ldots$. This is a contradiction, since there are now infinite strictly ascending chains of upward-closed sets, cf. [1], Lemma 3.4.

$\leq_n$ **is well-founded:** Assume that s is an infinite descending sequence on $\mathcal{D}^* \times \mathbb{N}$. Denote by s_1 the sequence of first components and by s_2 the sequence of second components, i.e., $s(i) = (s_1(i), s_2(i))$. Since $\sqsubseteq$ is a well-founded partial order, there is some j such that $s(k) = s(j)$ for all $k \geq j$. Thus, for $s(k) > s(\ell)$ for all $j \leq k < \ell$, which is impossible, since $\leq$ is well-founded.

$\leq_s$ **is a quasi-order:** Reflexivity is trivial. Consider $\mathbf{R}_1|Q_1 \leq_s \mathbf{R}_2|Q_2 \leq_s \mathbf{R}_3|Q_3$. By definition, $\mathbf{R}_1 \sqsubseteq \mathbf{R}_2 \sqsubset \mathbf{R}_3$, hence $\mathbf{R}_1 \sqsubseteq \mathbf{R}_3$. Additionally, due to the definition of $\sqsubseteq$, there is an N such that N = length $\mathbf{R}_1$ = length $\mathbf{R}_2$ = length $\mathbf{R}_3$.
There are three cases to consider:
1. $\mathbf{R}_1 = \mathbf{R}_2 = \mathbf{R}_3$. In this case, $\ell_{\text{length}(R_1)}(Q_1) \leq \ell_{\text{length}(R_2)}(Q_2) \leq \ell_{\text{length}(R_3)}(Q_3)$. By the above observation, this means that $\ell_N(Q_1) \leq \ell_N(Q_2) \leq \ell_N(Q_3)$, so $\ell_{\text{length}(R_1)}(Q_1) = \ell_N(Q_1) \leq \ell_N(Q_3) = \ell_{\text{length}(R_3)}(Q_3)$.
2. $\mathbf{R}_1 \neq \mathbf{R}_2 \neq \mathbf{R}_3$. Since $\sqsubseteq$ is a partial order, this implies in particular that $\mathbf{R}_1 \sqsubset \mathbf{R}_2 \sqsubset \mathbf{R}_3$, thus $\mathbf{R}_1 \sqsubset \mathbf{R}_3$ and hence $\mathbf{R}_1 \neq \mathbf{R}_3$.

3. $\mathbf{R}_1 \neq \mathbf{R}_2 = \mathbf{R}_3$ or $\mathbf{R}_1 = \mathbf{R}_2 \neq \mathbf{R}_3$. In either case, $\mathbf{R}_1 \neq \mathbf{R}_3$.

$\leq_s$ **is well-founded:** Let $\mathbf{R}_1|Q_1 \geq_s \mathbf{R}_2|Q_2 \geq_s \cdots$, and set $p_i := \phi(\mathbf{R}_i|Q_i)$. Then $p_1 \geq_n p_2 \geq_n \cdots$.
Since $\geq_n$ is well-founded, there is an i such that for all $j > i$, $p_j = p_{j+1}$. In particular, $p_j \not>_n p_{j+1}$. Thus, $\mathbf{R}_j \not>_s \mathbf{R}_{j+1}$ for all $j > i$.

□

Lemma 12. *If $\mathbf{R}|Q \mapsto \mathbf{R}'|Q'$ as a result of applying the* [Candidate]*,* [Decide]*,* [Conflict]*, or* [Induction] *rule, then $\mathbf{R}|Q >_s \mathbf{R}'|Q'$.*

Proof. Case analysis on the applied rule.

[Candidate]**:** In this case, $Q = \varnothing$, $\mathbf{R} = \mathbf{R}'$ and $Q' = \{\langle a, N\rangle\}$ for some $a \in \Sigma$. Thus, $\ell_N(Q) = N + 1 > \ell_N(Q') = N$.

[Decide]**:** In this case, $\min Q = \langle a, i\rangle$, $\min Q' = \langle b, i-1\rangle$ for some $a, b \in \Sigma$ and $i > 0$. Also, $\mathbf{R} = \mathbf{R}'$.
Thus, $\mathbf{R} = \mathbf{R}'$ and $\ell_N(Q) = i > i - 1 = \ell_N(Q')$.

[Conflict]**:** In this case, $\mathbf{R}' = \mathbf{R}[R_k^{\downarrow} \leftarrow R_k^{\downarrow} \setminus b\uparrow]_{k=1}^{i}$ for some $i \geq 1$, $b \in \mathrm{Gen}_i(a)$, $a \in \Sigma$. By definition of Gen, we have in particular that $b \in R_i^{\downarrow}$, and $b \notin R'^{\downarrow}_i$.
Since furthermore $R'^{\downarrow}_j \subseteq R_j^{\downarrow}$ for all $j \leq N$, we have $R'^{\downarrow} \sqsubset R^{\downarrow}$.

[Induction]**:** Analogous to [Conflict].

□

Proposition 2 *[Infinite sequence condition] For every infinite sequence* Init $\mapsto$ *$\sigma_1 \mapsto \sigma_2 \mapsto \cdots$, there are infinitely many i such that $\sigma_i \mapsto \sigma_{i+1}$ by applying the rule* [Unfold]*.*

Proof. Let Init $\mapsto \sigma_1 \mapsto \sigma_2 \mapsto \cdots$ be an infinite sequence of states. Since valid and invalid have no successor states, all σ_i must be of the form $\mathbf{R}_i|Q_i$. Thus, only the following rules can be applied to get from σ_i to σ_{i+1}: [Unfold], [Candidate], [Conflict], [Decide] and [Induction].

Suppose that there is some K such that for all $i > K$, the transition $\sigma_i \mapsto \sigma_{i+1}$ is not due to [Unfold].

But then, the transition is due to one of [Candidate], [Conflict], [Decide] and [Induction]. By Lemma 12, this means that $\sigma_K >_s \sigma_{K+1} >_s \sigma_{K+2} >_s \cdots$, i.e., from K on, the σ_i form a $>_s$-descending chain.

Since the σ_i form an infinite sequence, this implies that the sequence $\sigma_{K+0}, \sigma_{K+1}, \ldots$ forms an infinite $>_s$-chain. But by Lemma 3, $\leq_s$ is wellfounded, so no infinite $>_s$-chains exist – contradiction.

Thus, there must be infinitely many i such that $\sigma_i \mapsto \sigma_{i+1}$ using [Unfold]. □

Lemma 4 *If there is a path from I to $\Sigma \setminus P^{\downarrow}$ of length k, the rule* [Unfold] *can be applied at most k times: for every sequence* Init $\mapsto \sigma_1 \mapsto^* \sigma_n$*, there are at most k different values for i such that $\sigma_i \mapsto \sigma_{i+1}$ using the* [Unfold] *rule.*

Proof. Let Init $\mapsto \sigma_1 \mapsto^* \sigma_K$ be a sequence of rule applications in which has occured $N = k$ times, i.e., there are $i_1 < \cdots < i_k$ such that $\sigma_{i_j} \mapsto \sigma_{i_j+1}$ via [Unfold].

We wish to show that there is no σ' such that $\sigma_K \mapsto \sigma'$ via [Unfold].

If $\sigma_K \neq \mathbf{R}|Q$, the statement follows because valid and $\mathsf{invalid}$ have no successors. Thus, consider the case $\sigma_K = \mathbf{R}|Q$.

Let $s_0, \ldots, s_N$ be a path from I to $\Sigma \setminus P^{\downarrow}$, i.e., $s_0 \in I$, $s_k \in \Sigma \setminus P^{\downarrow}$ and $s_i \to s_{i+1}$ for $i = 0, \ldots, N-1$. Then, in particular, $s_i \in R_i^{\downarrow}$ for $i = 1, \ldots, N$ by (I2).

Thus, the pre-condition for [Unfold] is not fulfilled, since $s_i \in R_N^{\downarrow} \setminus P^{\downarrow}$. $\square$

Lemma 13. *For $i > L$, $D_i = \varnothing$. This implies that the set $\bigcup_{i \geq 0} D_i$ is finite.*

Proof. We first show a small auxillary fact:

Claim: $D_j \subseteq \mathsf{U}^{\uparrow}{}_j$ for all $j \geq 0$.
Proof: By induction on j.

- $D_0 \subseteq \Sigma \setminus P^{\downarrow} = \mathsf{U}^{\uparrow}{}_0$.
- $D_{j+1} \subseteq \mathrm{pre}(D_j\!\uparrow) \subseteq \mathrm{pre}(\mathsf{U}^{\uparrow}{}_j) \subseteq \mathsf{U}^{\uparrow}{}_{j+1}$, using the induction hypopthesis in the second step.

Now, let $i > L$. By Lemma 9, statement (2) and the above claim, we have $D_i \subseteq \mathsf{U}^{\uparrow}{}_i \setminus \mathsf{U}^{\uparrow}{}_{i-1} = \mathsf{U}^{\uparrow}{}_L \setminus \mathsf{U}^{\uparrow}{}_L = \varnothing$, since $\mathsf{U}^{\uparrow}{}_j = \mathsf{U}^{\uparrow}{}_L$ for all $j \geq L$ by Lemma 3.4 and the discussion in Paragraph 4 of [1].

Since for all $i > L$, $D_i = \varnothing$, it is sufficient to show that D_i is finite for $i = 0, ..., L$. This is guaranteed by Lemma 9, statement (5). $\square$

Lemma 6 *Let $\mathbf{a} \in \mathbb{N}^n$ be a state and $t = (\mathbf{g}, \mathbf{d}) \in \mathbb{N}^n \times \mathbb{Z}^n$ be a transition. Then $\mathbf{b} \in \mathrm{pre}(\mathbf{a}\!\uparrow)$ is a predecessor along t if and only if $\mathbf{b} \succeq \max(\mathbf{a} - \mathbf{d}, \mathbf{g})$.*

Proof. Suppose $\mathbf{b} \in \mathrm{pre}(\mathbf{a}\!\uparrow)$ is a predecessor along t. Then $\mathbf{b} \succeq \mathbf{g}$ and $\mathbf{b} + \mathbf{d} \succeq \mathbf{a}$. Thus, $\mathbf{b} \succeq \max(\mathbf{a} - \mathbf{d}, \mathbf{g})$. For the other direction, due to well-structuredness it is enough to show $\max(\mathbf{a} - \mathbf{d}, \mathbf{g})$ itself is a predecessor along t. But this holds since $\max(\mathbf{a} - \mathbf{d}, \mathbf{g}) \succeq \mathbf{g}$ and $\max(\mathbf{a} - \mathbf{d}, \mathbf{g}) + \mathbf{d} \succeq (\mathbf{a} - \mathbf{d}) + \mathbf{d} = \mathbf{a}$. $\square$

Lemma 7 *Let $\mathbf{a}, \mathbf{c} \in \mathbb{N}^n$ be states and $t = (\mathbf{g}, \mathbf{d}) \in \mathbb{N}^n \times \mathbb{Z}^n$ be a transition.*

1. *Let $\mathbf{c} \preceq \max(\mathbf{a} - \mathbf{d}, \mathbf{g})$. Define $\mathbf{a}'' \in \mathbb{N}^n$ by $a''_j := c_j + d_j$ if $g_j < c_j$ and $a''_j := 0$ if $g_j \geqslant c_j$, for $j = 1, \ldots, n$. Then $\mathbf{a}'' \preceq \mathbf{a}$. Additionally, for each $\mathbf{a}' \in \mathbb{N}^n$ such that $\mathbf{a}'' \preceq \mathbf{a}' \preceq \mathbf{a}$, we have $\mathbf{c} \preceq \max(\mathbf{a}' - \mathbf{d}, \mathbf{g})$.*
2. *If $\mathbf{a} \preceq \max(\mathbf{a} - \mathbf{d}, \mathbf{g})$, then for each $\mathbf{a}' \in \mathbb{N}^n$ such that $\mathbf{a}' \preceq \mathbf{a}$, it holds that $\mathbf{a}' \preceq \max(\mathbf{a}' - \mathbf{d}, \mathbf{g})$.*

Proof. For the first part, consider coordinate j for $1 \leqslant j \leqslant n$. If $g_j \geqslant c_j$, then $a''_j = 0 \leqslant a_j$ and $c_j \leqslant g_j \leqslant \max(a'_j - d_j, g_j)$.

On the other hand, suppose $g_j < c_j$. First note that $\max(a_j - d_j, g_j) = a_j - d_j$ since $g_j < c_j \leqslant \max(a_j - d_j, g_j)$. Thus,

$$a''_j = c_j + d_j \leqslant \max(a_j - d_j, g_j) + d_j = (a_j - d_j) + d_j = a_j$$

and

$$c_j = a''_j - d_j \leqslant a'_j - d_j \leqslant \max(a'_j - d_j, g_j)\,.$$

For part (2), consider coordinate j for $1 \leqslant j \leqslant n$. If $a_j - d_j \geqslant g_j$, then

$$a_j \leqslant \max(a_j - d_j, g_j) = a_j - d_j \,.$$

Therefore $d_j \leqslant 0$, implying

$$a'_j \leqslant a'_j - d_j \leqslant \max(a'_j - d_j, g_j)\,.$$

On the other hand, if $a_j - d_j < g_j$, then

$$a'_j \leqslant a_j \leqslant \max(a_j - d_j, g_j) = g_j \leqslant \max(a'_j - d_j, g_j)\,.$$

$\square$